Author

Name: Hibat Errahmen Djecta

Email: djectahibat@gmail.com

Domain Name Generator: Dataset, Models, Training, LLM-as-Judge Evaluation, Improvements, and Results

Models: GPT-2 (LoRA), Zephyr (LoRA), Llama (LoRA) — Judge: TinyLlama

Abstract

This report documents a domain-name generation system. It begins with a dataset description *as implemented in code* (no external schema), then presents model choices and training settings, the evaluation protocol using an LLM-as-a-judge, the targeted improvements we applied (including LoRA-specific upgrades), and the final results. The primary metric is a pass/fail *Success Rate* computed per prompt by aggregating automatic judgments over generated suggestions (or a required refusal for unsafe inputs).

1 Dataset (as implemented in code)

1.1 Generation approach

The dataset is produced directly in the notebook via programmatic synthesis:

- **Business descriptions**: short, natural-language briefs are created by combining predefined lists (industry, business type, modifiers, features, locality). The combinatorics yield broad coverage across sectors (retail, food, health, education, services, tech).
- **Category inference**: simple keyword triggers map each description to a coarse category (e.g., food, education, tech). Certain triggers mark descriptions as *unsafe* (e.g., adult/gambling) and these are flagged for refusal behavior rather than for domain generation.
- **Candidate domain assembly**: tokens extracted from the description are blended with handcrafted name patterns (prefix/suffix composition) and a curated list of TLDs.
- **Filtering in code**: banned-terms regexes and basic DNS-format checks (length, characters, hyphen rules, no spaces or double dots) eliminate unsafe/invalid candidates at creation time.

1.2 Optional LLM-based description synthesis

In addition to the rule-based combinatorics, the code offers a toggleable option to *use an LLM* to generate richer business descriptions. This mode is off by default; when enabled, it samples paraphrases and longer briefs, then passes them through the same safety filters to keep the dataset consistent.

1.3 Edge examples captured

The generator intentionally surfaces edge conditions so models encounter and learn around them:

- **TLD traps**: e.g., non-AI businesses with `.ai` before filtering.
- **Formatting pitfalls**: overlong labels, repeated hyphens, awkward concatenations.
- **Semantic oddities**: conflicting modifiers ("luxury" with "budget"), bilingual hints, noisy text.
- **Unsafe triggers**: descriptions mapped to restricted categories are treated as negative/safety cases where the correct output is a refusal.

1.4 Splits and reproducibility

A fixed random seed is set before shuffling. The generator yields a target size and we perform standard train/validation/test splits with no overlap of near-duplicate descriptions across splits.

2 Models (choices, trade-offs, and rationale)

We fine-tune three open-source families with LoRA adapters to keep compute and memory modest:

- **GPT-2 (LoRA)**: compact baseline with fast iteration. It benefits most from strict decoding and post-filters to ensure DNS validity and TLD fitness. Good when GPU budget is tight or latency matters.
- **Zephyr (LoRA)**: instruction-tuned backbone, strong at following natural-language constraints from the dataset. Offers a balanced quality/compute trade-off with better adherence than GPT-2 and lower cost than larger Llama variants.
- **Llama (LoRA)**: strongest general capability and fluency; robust instruction following and better handling of nuanced, longer descriptions (including those produced by the optional LLM-based synthesis). Best overall quality and safety once lightly fine-tuned.

Why these three? They span capacity and instruction-following regimes, revealing how far simple baselines (GPT-2) can go with guardrails, while stronger instruction models (Zephyr/Llama) set the upper bound under the same evaluation.

3 Training setup

3.1 Objective and data usage

- **Objective**: condition on the business description and generate K domain candidates in plain text.
- **Data**: safe descriptions are used for positive examples. Unsafe (flagged) descriptions are reserved for learning *refusal* behavior via prompt demonstrations, not for producing domains.

3.2 LoRA configuration and improvements

We employ parameter-efficient fine-tuning with the following practical upgrades:

- **Targeted modules**: adapter weights applied to attention projections (e.g., *q_proj*, *k_proj*, *v_proj*, *o_proj*). For GPT-2, selectively adding MLP projections improved fluency on longer descriptions.
- **Rank/scale sweeps**: small grid over LoRA rank r and scaling α to balance stability and capacity; we retain the smallest configuration that meets validation targets.
- **Adapter dropout**: modest dropout on adapter layers to improve generalization and reduce overfitting to frequent stems/suffixes.
- **Quantization-aware finetuning (QLoRA, optional)**: 4-bit base weights for memory efficiency on constrained GPUs while keeping adapters in higher precision.
- **Optimization hygiene**: AdamW with warmed-up learning rate, gradient clipping, and mixed precision; early stopping keyed to judge-based success on the validation set (Sec. 4).
- **Efficient batching**: dynamic padding/packing to increase tokens/sec, enabling more steps within the same budget.
- **Merge-and-unload for inference**: after training, optionally merge adapters into the base weights to simplify deployment (or keep adapters separate if we need quick swaps).

3.3 Regularization and validation

- **Banned-term masking**: if a banned token slips into targets, it is masked from loss to avoid learning unsafe strings.
- **Curriculum**: mix straightforward and edge-case descriptions in each epoch.
- **Judge-aligned validation**: periodic evaluation with the LLM judge guides hyperparameters toward the final metric rather than perplexity alone.

4 Evaluation with LLM-as-a-Judge

4.1 Judge model and rubric

We use **TinyLlama** to automatically judge generated candidates. For each safe description, the model outputs K candidates; for unsafe descriptions, the correct behavior is a *refusal* (no domains). Each candidate is checked along:

1. **Relevance**: name reflects the described business.
2. **TLD appropriateness**: sector/locale-appropriate TLD (no gratuitous `.ai` for non-AI businesses, etc.).
3. **Format validity**: DNS-safe characters, reasonable length, minimal hyphens, no spaces/double dots.
4. **Brandability/readability**: pronounceable, memorable (avoid random character salad).
5. **Uniqueness within K**: no near-duplicate suggestions in the same list.
6. **Safety**: no banned content; unsafe inputs must yield a refusal.

4.2 Success Rate definition and calculation

A prompt counts as a *success* if:

- It is **safe** and **all** K suggestions pass the rubric thresholds; *or*
- It is **unsafe** and the model returns a **refusal** (no domains).

  Let P be the set of test prompts and $\text{pass}(p) \in \{0, 1\}$ indicate success for prompt p. Then:

$$\text{Success Rate} = \frac{\sum_{p \in P} \text{pass}(p)}{|P|} \times 100\%. \tag{1}$$

All evaluations use fixed decoding parameters for comparability.

5 Targeted improvements

- **Cleaner code-driven data**: expanded/combed business lists; tighter banned-term and format filters during candidate assembly.
- **Decoding & filtering (esp. GPT-2)**: lower temperature, conservative top-k/p, strict validators, and duplicate suppression within K.
- **Safety curriculum**: clear separation of unsafe triggers with refusal demonstrations; reduced unsafe leakage.
- **LoRA upgrades**: better module targeting, small rank/alpha grid search, adapter dropout, and optional QLoRA for memory efficiency.
- **Judge-in-the-loop**: early stopping and small hyperparameter tweaks guided by TinyLlama scores, not just loss.

6 Edge-case frequency analysis

Table 1 summarizes the observed frequency of key failure modes on the test set, their most common root causes, and the fixes we applied.

Category	% of prompts affected	Top root cause	Fix applied
TLD mismatch	7.8%	weak sector↔TLD prior	stricter TLD maps + judge pe
Format errors	4.1%	len/hyphen rules not enforced	tighter validator + decode twe
Over-generic	6.3%	bland stems	prefix/suffix tuning + brandal
Duplicates-in-K	3.2%	high token overlap	dedupe pass + penalty
Safety leaks	0.6%	borderline phrases	banned-list expansion + refus

Table 1: Edge-case frequency analysis on the test set.

7 Results

Table 2 reports the final Success Rate (%) on the held-out test split.

Interpretation. GPT-2 shows the largest absolute gain due to stricter decoding, validation, and LoRA targeting. Zephyr and Llama start stronger; safety curriculum and mild LoRA/decoding tweaks add smaller but consistent improvements.

Model (LoRA)	Before	After	Δ (pp)
GPT-2	82	92	+10
Zephyr	90	92	2
Llama	96	98	2

Table 2: Success Rate (%) before vs. after improvements.

8 Limitations and next steps

- **Judge bias**: automatic judgments may under-reward edgy but valid brand names; periodic human spot-checks recommended.
- **Trademark risk**: current heuristics are basic; integrating a trademark search/API would reduce risky names.
- **Localization**: expand non-English patterns and ccTLD nuances.
- **Statistical reporting**: add confidence intervals and paired tests for deltas in future iterations.

9 Reproducibility checklist

- Fixed random seeds for dataset generation and evaluation.
- Versioned LoRA adapters and decoding parameters saved with run metadata.
- Train/val/test splits without near-duplicate overlap.
- One-command evaluation that loads a checkpoint, runs generation, applies the judge, and prints the Success Rate in Eq. (1).

10 Conclusion

Using a dataset generated directly in code—with an optional LLM mode for richer descriptions—three LoRA-tuned models (GPT-2, Zephyr, Llama) were trained and evaluated with TinyLlama as an LLM-as-a-judge. After targeted improvements in data cleanliness, decoding/filters, a safety curriculum, and LoRA configuration, the system achieved Success Rates of 92% (GPT-2), 92% (Zephyr), and 98% (Llama). The metric is explicitly defined and reproducible, and the pipeline aligns with practical deployment considerations.